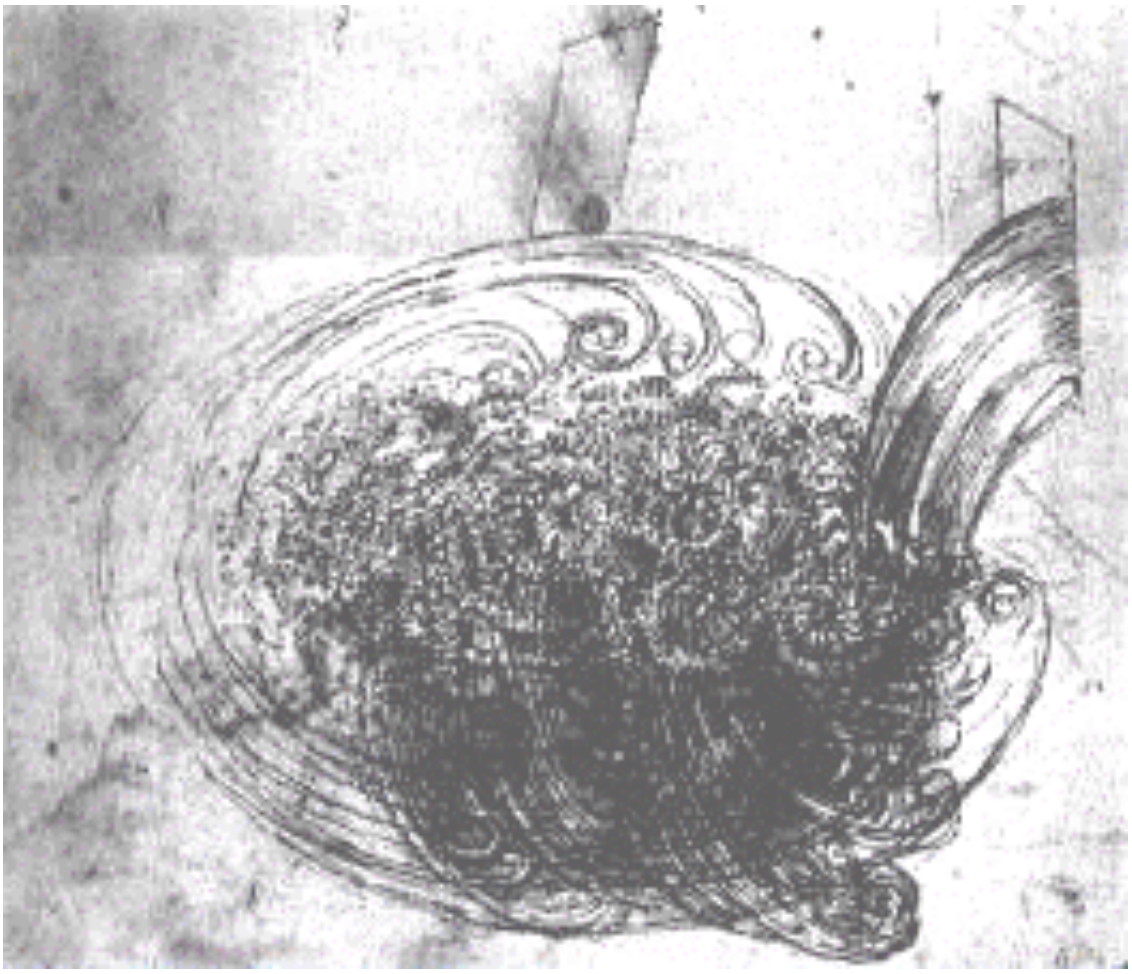
User's Guide

Version 4.4
December 2013



CryoSoft

FLOWER

Hydraulic Network Simulation

DISCLAIMER

Even though CryoSoft has carefully reviewed this manual, CRYOSOFT MAKES NO WARRANTY, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS PROVIDED "AS IS", AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL CRYOSOFT BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

Contents

Roadmap

Before you start

This manual is the reference user's guide for FLOWER and its post-processor, FLOWERPOST. Throughout this manual we assume that the reader is familiar with the physics and engineering issues that are associated with the design and analysis of a cryogenic pipework consisting of both passive (pipes, heat exchangers, valves) and active (pumps, turbines) components. Details on the physics modeling that is at the basis of the program are given in [1], [2] and [3]. We strongly suggest that the reader consults and familiarizes with these references before using this manual.

How to use this manual

This manual is structured as follows:

- Chapter 1 contains a brief and general introduction on the modeling principle and solution methods available.
- Chapter 2 gives basic information on the installation, explains how to start a FLOWER run and launch the post-processor FLOWERPOST on a UNIX workstation.
- Chapter 3 contains case studies that the reader should use to familiarize with the operation and features of the program.
- Chapter 4 contains additional information on the preparation of the input and the meaning of the input variables
- Chapter 5 describes the details of the post-processing command language.
- Chapter 6 describes the External Routines that can be used for advanced use. These routines can be linked to the standard code to provide powerful customization.
- Chapter 7 deals with troubleshooting and error messages;
- Chapter 8 gives the references and a general bibliography for documentation.

Beginners to FLOWER should read chapters 1, 2 and 3 in sequence. They will make occasional cross-reference to chapters 4 and 5 for detailed information. Experienced users will use chapters 4, 5 and 6 for daily operation. Chapter 7 is designed as an indexed glossary for error messages and associated actions.

CHAPTER 1

Introduction

What is FLOWER

FLOWER is a model for the thermo-hydraulic simulation of an arbitrary, user defined assembly of manifolds, pipes, heat exchangers, valves, turbines and pumps, the *hydraulic network*. It provides pressure, temperature and massflow conditions throughout the network and as a function of time. It is aimed at the simulation of the integrated response of the proximity cryogenics of a whole installation, including the pipework of a magnetic system.

A FLOWER model

A typical cryogenic system can be regarded in first approximation as an assembly of active and passive components forming an hydraulic network. In a FLOWER model we consider the network as composed of:

- interconnected junctions where the flow can be steady state or transient. Junctions can be of different type, passive (e.g. a pipe or a valve) or active (e.g. a pump or a turbine);
- volume nodes with perfect mixing of helium and zero flow, representing buffers and manifolds.

In the model definition the junctions always interconnect volume nodes. Volumes, however, can have negligible size so that the result is a direct interconnection of two (or more) junctions in the same point. The junction definitions are based on the following types included in the model:

- 1-D flow pipes, including full compressible flow and propagation delay and waves,
- valves, with concentrated head loss and isenthalpic flow,
- pumps (volumetric or centrifugal), with isentropic flow,
- turbines, also with isentropic flow.

In addition the model considers thermal links among 1-D flow pipes, as e.g. in the case of heat exchange between parallel flows, among 1-D pipes and volumes, as would be the case for heat exchange between a pipe submerged in a bath, and among volumes. Figure 1 shows a schematic representation of a possible hydraulic network. We refer to [3] for the details of the model implemented. Here we recall only the main assumptions and limitations that affect the user:

- the flow of helium is in single phase, but can be both in supercritical or superfluid state. Superfluid counter-flow heat exchange is automatically activated when the state is below the lambda transition;
- all volumes/manifolds are assumed to have perfect mixing of the incoming flows, and are therefore identified by a single pressure and temperature;
- the flow in all connections (pipes, valves, pumps and turbines) is assumed to be compressible but steady state, i.e. all wave propagation delays are neglected through these components. The only exception is the compressible flow pipe element that model the flow using the full set of compressible flow equations;
- pumps are always providing a massflow, either independent on the pressure difference from inlet to outlet as is the case for volumetric pumps, or depending on the pressure difference as is the case for compressors. In both cases the pump is assumed to act ideally, i.e. the fluid undergoes an isentropic transformation from inlet to outlet;
- turbines act on the flow as concentrated hydraulic resistances in which the fluid undergoes an isentropic state transformation.

With these hypotheses the network can be solved using mass and energy balances at the volumes/manifolds and momentum and energy balances in steady state in the incompressible flow connections. The compressible flow pipe is based on the complete solution of transient mass, momentum and energy balances. Details on the model and a description of the equations solved are given elsewhere [2,3].

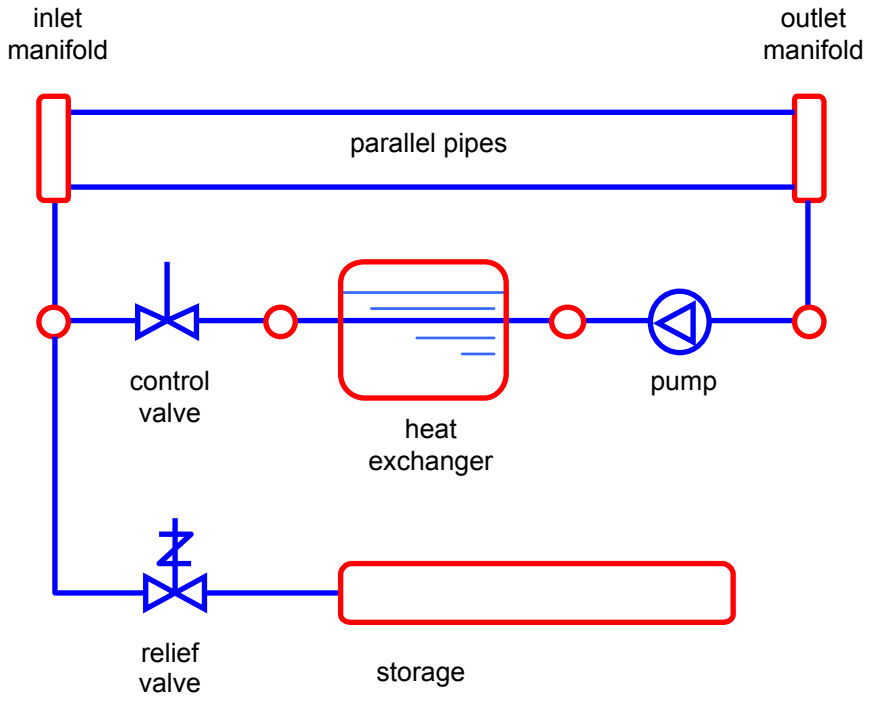


Figure 1. Schematic view of an hydraulic network as can be modeled and solved by the network solver (FLOWER). The nomenclature is indicated for clarity. In this example two parallel pipes are connected through manifolds to a controlled pump and heat exchanger. Any pressure excess in the loop is accommodated discharging through the relief valve in the storage.

Model Solution

The equation for the network of hydraulic components in FLOWER are detailed in [2, 3]. The solution is based on the use of finite elements in space and a time stepping algorithm. The volumes in the network are the nodes in the mesh, where mass and energy balances are written in discrete form (note that velocity is assumed to be zero in the volumes, and hence the momentum balance is not used). Most hydraulic junctions (steady-state flow pipes, valves, pumps and compressors, turbines) are discretized as a single finite element, with mass, momentum and energy balances written at the inlet and outlet nodes, which correspond to volumes connected by the junction in the network. Compressible flow pipes are treated similarly, except for the fact that they have an arbitrary number of internal nodes.

Once the discretization in space is achieved as described above, the resulting equations form a system of ordinary differential equations in time. The solution of this system is performed in FLOWER using an implicit time stepping algorithm, which can be selected by the user to be either of first or second order accuracy. FLOWER advances the time integration by automatically adapting the time step based on a tolerance provided by the user, various error control criteria that can be selected from input, and maximum and minimum time step bounds. The adaptive stepping automatically limits the change of the system variables (pressure, temperature and velocity) and thus achieves accurate and stable results in the presence of the non-linearity inherent to fluid flow.

Post-processing

The results produced by FLOWER are integrally stored and can be analyzed to produce plots and reports by the post-processor FLOWERPOST. FLOWERPOST responds to a user-friendly command language and allows selection of results in time or space, plot and print-out of results vs. time or space, parametric plot of results at given time or space coordinate. See the case studies in Chapter 3 for examples of post-processing sessions, and Chapter 5 for the details on the syntax of the command language.

User Flexibility and Further Extensions

FLOWER has several features that allow customizing its modeling capability beyond the allowable parameterization of the thermal/hydraulic/electric configuration that can be achieved using the standard input file. Specifically, the user can:

- modify the dependence of geometry, waveforms and material properties on space, time and solution variables, beyond the standard models implemented, using *External Routines* that can be statically linked to the program segments through a compilation step that produces a customized version of the code. See Chapter 6 for documentation on *External Routines*;
- change parametrically the behavior of the *External Routines* by making use of *Variables* that are read by the code input parser, and can be accessed at run-time using the *Variables* library. See Chapter 4 for details on the syntax to be adopted for the *Variables* input block;
- couple to other programs of the CryoSoft suite through the multi-tasking code manager SUPERMAGNET. This allows to augment the physics span of the simulation domain to include thermal networks (e.g. heat exchange in a coil), hydraulic networks (e.g. proximity cryogenics) or electrical circuits (e.g. magnet protection).

CHAPTER 2

Installing and Running FLOWER

Platforms

FLOWER and its post-processor FLOWERPOST are provided as a package developed for running under UNIX or UNIX-like (e.g. Linux) operating system. The reason is that they require computer intensive calculations, orderly file management and little interactivity. At the time when this manual is written, the platform where FLOWER has been developed is

- Macintosh running MacOS-X (10.8.5 and higher) under XQuartz,(2.7.4) gcc (4.8.1) with gfortran.

The code has been installed and tested on the following platforms:

- IBM-RISC workstations running the AIX OS;
- Sun-microsystem workstations running the Solaris OS;
- DEC-alpha workstations running OSF1 OS;
- HP workstations running HP-UX OS;
- INTEL PC's running RedHat Linux OS;
- Linux boxes running on Scientific Linux (SLC dialect).

Although UNIX obeys strict standards, the architecture of the operating and file system may vary from vendor to vendor. It is therefore possible that porting may require minor adaption of code and libraries. Contact us for advice.

In the following sections we assume here that you are running under a UNIX or UNIX-like operating system, and that you are familiar with UNIX commands, directory and file handling. Contact your system administrator for matters regarding UNIX commands and file system.

Although versions of FLOWER and FLOWERPOST have been ported to PC's running the Windows OS, at the time when this manual is written this is not a platform directly supported and part of the instructions provided below (i.e. how to run and post-process a case) may not be directly applicable.

Installation

FLOWER is one of the CryoSoft family of programs. You will have therefore received the CryoSoft package containing FLOWER either as a tar-ball or in pre-installed form. Verify in the CryoSoft installation manual [4] the procedure to be followed for the proper installation of the complete package. The executable codes, `flower` and `flowerpost` are in the directory

`~/CryoSoft/bin/`. You will find the example inputs and post-processing command files in the directory `~/CryoSoft/xample/flower/code_x.x/` (the symbol `~/` stands for your home directory, `x.x` is the version you received).

How to run FLOWER

Start-up To run FLOWER you will need to launch the executable code. In the standard installation on a UNIX system described above FLOWER is launched typing the command:

```
~/CryoSoft/bin/flower
```

Once launched, the program prompts the user for the input file name. FLOWER reads the problem definition from an ASCII file whose structure and content are described in detail in Chapter 4 of this manual. Examples of input files are given in Chapter 3. At this time you will enter the name of a file containing the input for the case to be run (e.g. `file.input`):

```
Enter input file name
file.input
```

FLOWER then parses the input file, performs checks on consistency, configures the case and starts the simulation. A simulation starts from an initial condition at the starting time and advances in time using the time step selected. At each time step FLOWER emits a message with the real time reached in the simulation (in s) the time step taken (in s) and the ratio of real time to the total time to be simulated:

```
....
Time : 4.949E-03   Step :  3.235E-05   Time/Tend :   0.98987
Time : 4.998E-03   Step :  4.852E-05   Time/Tend :   0.99957
....
```

until the end of the simulation. When the end time of the simulation is reached FLOWER prints a message reporting the total CPU time used in the run:

```
Total Cpu [s]:   244.059998
```

Each run of FLOWER produces:

- a binary storage file containing all results stored at user's specified times. The user can control the name of this file, the default file name is `flower.store`;
- a log file containing a report on the case run, run statistics and error messages. The user can control the name of this file; the default file name is `flower.log`.

Restart After a successful completion of a run it is possible to restart the simulation at the last time stored in the binary storage file and proceed with the time integration. A restart procedure is triggered if the input file read by FLOWER contains the `Restart` command (see Chapter 3 and 4 for details). Assuming that this is the case for the input file `file.restart`, a restart in our example is obtained launching again FLOWER:

```
~/CryoSoft/bin/flower
Enter input file name
file.restart
```

in which case FLOWER reads the binary storage file and starts the simulation at the last time stored:

```
Time : 5.000E-03   Step :  1.000E-05   Time/Tend :   0.00000
```

Until the final time specified in the input file `file.restart` is reached.

Note You can use an arbitrary sequence of restarts to simulate different time spans with varying resolution and accuracy. There is no limit to the number of restarts that can be executed for a single simulation.

How to run FLOWERPOST

To produce any detailed result, both in the form of printed tables or plotted curves in PostScript® format, it is necessary to run the FLOWER post-processor FLOWERPOST. FLOWERPOST is launched under UNIX with the command:

```
~/CryoSoft/bin/flowerpost
```

FLOWERPOST prompts the user for the name of an ASCII file containing the series of commands that control the generation of the printouts and plots. The structure and content of this file is described in detail in Chapter 5 of this manual. Examples of command files are given in Chapter 3. At this time you will enter the name of the file containing the commands (e.g. `file.post`):

```
Enter command file name
file.post
```

FLOWERPOST then parses, echoes and interprets the commands from the command file. The commands cause retrieval of the results of a run from the binary storage file generated by FLOWER (by default from the file `flower.store`). As a result FLOWERPOST generates:

- a file containing the formatted printouts of the results (by default `flowerpost.out`), and
- a file containing the plots requested in PostScript® format (by default `flowerpost.ps`).

Customization

The method described earlier provides the standard manner to run a FLOWER simulation, and post-process the results. FLOWER, however, as most other CryoSoft codes, gives the possibility to customize the physical models by using External Routines, as described in Chapter 6 (see later for details). The user has the possibility to adapt and extend the physics contained in the standard solver, at the additional complexity of writing FORTRAN routines that must obey to the language syntax, and parameter call specification. The customized External Routines need to be compiled and linked the program segments to generate the customized version of the code. Template for the External Routines are given in the directory `~/CryoSoft/usr/flower/code_x.x`. Compilation and link-editing can be done using the standard installation script `CSmake`, but we discourage users to modify the standard codes provided, as this will replace the reference installation. As a safer alternative, we strongly recommend copying the External Routines templates in a work directory, and generating in this location the customized version of the code by using an adapted compilation script, or a makefile. Consult the examples below, and contact us for guidelines on how to set-up one such customized structure.

CHAPTER 3

Case Studies

As discussed in Chapter 2, FLOWER requires an input file with all definitions necessary to specify the assembly of components in the model structure, the characteristics of each component, the initial conditions, and the solution controls. We refer to this file as the *input file*. The input file is needed both for a start-up run and a restart run.

Similarly, post-processing of FLOWER results using the post-processor FLOWERPOST requires an input file with a sequence of commands that select results, print and plot them. We refer to this file as the *post-processing command file*.

In this Chapter we give examples of input files and post-processing command files to deal with practical modeling situations. The case studies given here are intended to guide the user from the formulation of a problem to its modeling, the creation of the input file for the case, running the case, and finally the generation of the results. For obvious reasons, they are of limited complexity and are intended as examples to illustrate minimum capability of the program. More complex situations can obviously be modeled, taking the following case studies as starting points and evolving or combining them. Refer to Chapter 2 on how to run the examples described here with FLOWER and how to generate results and plots with FLOWERPOST.

Note All input files and post-processing command files for the case studies discussed in this manual are provided with the standard installation. They are located in the directory:

`~/CryoSoft/xample/flower/code_x.x`

(where `x.x` stands for the version you received). In the following sections we use the `Courier` font to reproduce the content of the input files, while text in *italic* (starting with a semicolon ";" in the input files) indicates comments to the input that are ignored by the input parser.

Parallel flow heat exchanger

Physical definition of the problem This case shows how to model heat exchange between two pipes where helium is flowing at different temperatures. The direction of the flow, inlet to outlet, is the same for the two pipes. The flow is driven by a pressure difference of 0.05 bar between the inlet and outlet manifolds of the pipes. These manifolds are assumed to have a very large (ideally infinite) volumes.

The inlet temperature for the first pipe is 300 K, while for the second pipe it is 320 K. Heat exchange takes place between the two pipes through a thermal resistance that is obtained as the series of the following thermal resistances:

- thermal resistance associated with heat convection between the helium flow in the first pipe and the pipe wall,
- equivalent thermal resistance of the pipe wall and any intermediate thermal barrier, as specified in the thermal link definition (the value assumed is 0.1 K/W m distributed along the whole length of the pipes), and
- thermal resistance associated with heat convection between the pipe wall and the helium flow in the second pipe.

Note that while the thermal resistance of the pipe walls is given in input, the heat transfer from the flow to the pipe wall through the thermal boundary layer is computed automatically for the two flows.

As shown below the two pipes have a length of 10 m and an internal diameter of 1 cm. The friction factor and heat transfer coefficient for the flow are obtained from the standard correlations.

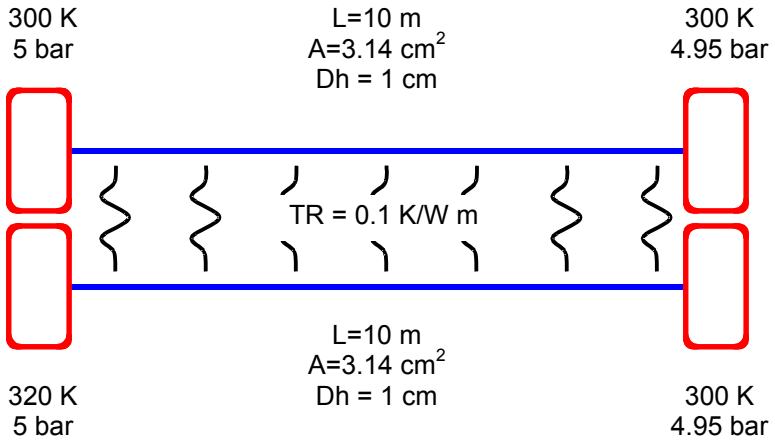


Figure 2. Schematic view of the model for the analysis of the temperature in the two pipes forming a parallel current heat exchanger. The thermal resistance (indicated with TR) represents only the contribution of the pipe walls and possibly any intermediate thermal barrier, but does not include the heat transfer from the flow to the pipe wall through the thermal boundary layer. This last is computed automatically for the two flows.

Input file for the start-up run The model for this case requires the definition of four volumes (the manifolds), the two pipes connecting them, and the thermal link between the pipes. The step-by-step definition of the input file for `FLOWER` start-up run is shown below.

```
;
; Simulation of a parallel current heat exchanger formed by two
; pipes thermally linked. Run this file to generate a binary storage
; for later post-processing with co-currentHX.post
;

Begin Simulation
   title         co-currentHX

; The total number of volumes, junctions and links in the model defined
; subsequently must be given before reading the definition of each
; component

   Volumes       4
   Junctions     2
   Links   1

; The time integration starts at StartTime and ends at EndTime, with
; binary output of the results every OuputStep

   StartTime     0.0
   EndTime       25.0
   OutputStep    0.1

; Definition of the time stepping algorithm, minimum and maximum steps,
; error control strategy for the time integration, and tolerance to be
; achieved

   TimeMethod    EulerBackward
   MinimumStep   1.0e-6
   MaximumStep   1.0
   StepEstimate  smooth
   ErrorEstimate change
   ErrorControl  on
   Tolerance     1.0e-6

; Log output is directed to the file co-currentHX.log, while results are
; stored in the file co-currentHX .store for later restart and reporting

   StorageFile  co-currentHX.store
   LogFile      co-currentHX.log
End

; Here is the definition of the inlet and outlet volumes for the first
; pipe (junction 1 below). Note the large volume assumed to make the
; manifold equivalent to an infinite reservoir at constant pressure and
; temperature

Begin Volume 1  ; inlet volume node
   V 1.0e6  P 5e5   T 300.0
End

Begin Volume 2  ; outlet volume node
   V 1.0e6  P 4.95e5   T 300.0
End

; Inlet and outlet volumes for the second pipe (junction 2 below).
; The inlet manifold for this pipe has an higher temperature than for
```

```
; the first pipe, which will result in heat exchange between the two flows

Begin Volume 3  ; inlet volume node
   V 1.0e6  P 5e5   T 320.0
End

Begin Volume 4  ; outlet volume node
   V 1.0e6  P 4.95e5   T 300.0
End

; Definition of the first pipe, connecting volumes 1 and 2 defined
; previously. The wetted perimeter enters in the definition of the total
; thermal resistance from the flow in pipe 1 to pipe 2, and is therefore
; required when linking junctions to other junctions (or volumes)

Begin Junction 1
   type Cpipe
   connection 1 2
   L 10.0   A 3.14e-4   Dh 1.0e-2   N  300
   WP 3.14e-2
End

; The second pipe, connecting volumes 3 and 4, has the same
; characteristics as pipe 1

Begin Junction 2
   type Cpipe
   connection 3 4
   L 10.0   A 3.14e-4   Dh 1.0e-2   N  300
   WP 3.14e-2
End

; The link block defines the thermal connections of elemen. In this case
; the thermal link is between junctions (JJ type), linking pipes 1 and 2

Begin Link 1
   type JJ
   connection 1 2
   ThermalResistance 0.1
End

; At this point the input definition is complete and execution starts
```

Input file for the restart run To proceed with the simulation for a longer time than 25 s (the `EndTime` specified in the start-up run) we use the restart feature of FLOWER. Below we give the step-by-step definition of the input file for the restart of the simulation with a reduced time resolution in the storage of results.

`co-currentHX.restart`

```
; In case of restart only the simulation parameters are needed. All
; other parameters are taken from the storage file generated during
; the previous run

Begin Simulation

; The presence of the Restart keyword is necessary to trigger a restart run

  Restart
```

```
; The time integration starts at the last time stored on file co-currentHX.store
; (as specified below) and proceeds to the new EndTime, with the prescribed
; OutputStep

  EndTime      50.0
  OutputStep    5.0

; Log output and results are appended to the existing files during the restart

  LogFile       co-currentHX.log
  StorageFile   co-currentHX.store

end
```

Post-processing command file The following is an example of the sequence of commands necessary to generate of print-outs and plots using the post-processor FLOWERPOST.

co-currentHX.post

```
;
; Post-processing sequence for the parallel current heat exchanger
; simulation. A PostScript output file co-currentHX.ps is generated
; containing all plots.
;

; Define the file where results are stored

StorageFile    co-currentHX.store

; Define the file for Postscript(R) and printed output

PostScriptFile co-currentHX.ps
OutputFile     co-currentHX.out

; The number of plots per page can be set to 1, 2, 3, 4 or 6

set plotsperpage 3

; Select the results of the simulation at the times closest to those below,
; all following plots are as f(x), the selected times are parameters

select time 0.1 1 10

; Plot various quantities as f(x) selecting the quantity first, the component(s)
; next

plot temperature junction 1 junction 2
plot pressure junction 1 junction 2
plot velocity junction 1 junction 2

; The stop command terminates parsing, the post-processing session is finished

stop
```

Results The PostScript file output `co-currentHX.ps` is generated running the post-processor FLOWERPOST with the commands described above in the file `co-currentHX.post`.

Note You will need a PostScript viewer to look at the plots in the PostScript file. The standard viewer, usually installed on UNIX systems, is `gs`. Try to launch the viewer with the command:

```
gs co-currentHX.ps
```

The plots below show the first page in the PostScript output `co-currentHX.ps`. As requested in the commands file, the three plots are the temperature, pressure and velocity distributions in the junctions 1 and 2 at selected times.

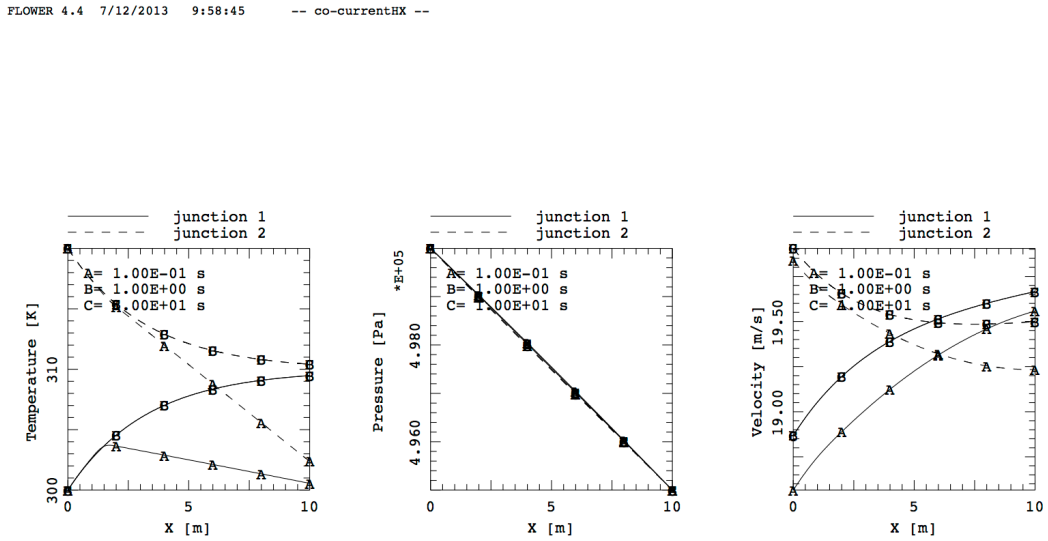


Figure 3. Temperature, pressure and flow velocity along the two pipes of the parallel flow heat exchanger at different times, showing steady state conditions reached after 1 s from the beginning of the simulation.

Counter flow heat exchanger

Physical definition of the problem In this test case we consider a counter-flow heat exchanger with the same characteristics taken for the parallel flow heat exchanger described in the previosu section. The topology for this case is shown in Fig. 4. Note that this is nearly identical to Fig. 2, apart for the different pressure and temperature in the two volumes that model the manifolds for the second pipe. Having chosen a higher pressure at outlet results in backwards flow in this pipe. The value of the thermal resistance is also modified to a lower value of 0.05 K/W m.

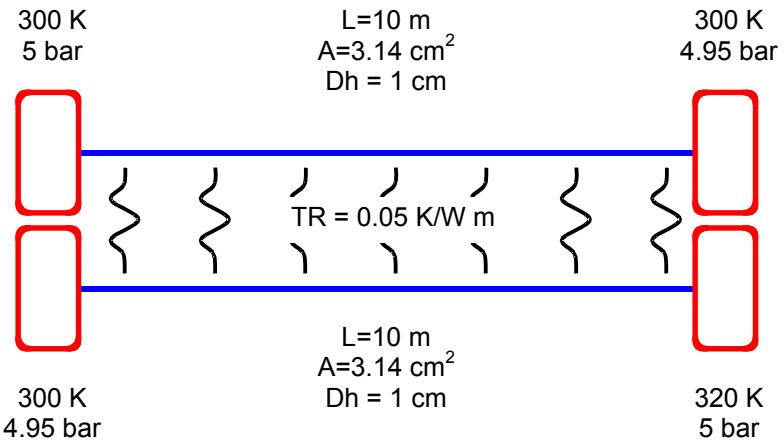


Figure 4. Schematic view of the model for the analysis of the temperature in the two pipes forming a counter current heat exchanger. The characteristics are identical to those reported in Fig. 2 apart for the values of the pressures in the manifolds of the second pipe.

Input file for the run The input for this run requires the definition of the same number of components as for the parallel flow heat exchanger (4 volumes, 2 pipes and 1 thermal link). The step-by-step definition of the input file is shown below.

counter-currentHX.input

```
;
; Simulation of a counter-current heat exchanger formed by two
; pipes thermally linked. Run this file to generate a binary storage
; for later post-processing with counter-currentHX.post
;

Begin Simulation
   title          counter-currentHX

   Volumes     4
   Junctions   2
   Links       1

   StartTime   0.0
   EndTime     10.0
   OutputStep  0.1

   TimeMethod  EulerBackward
```

```
   MinimumStep       1.0e-3
   MaximumStep       1.0
   StepEstimate      smooth
   ErrorEstimate change
   ErrorControl      on
   Tolerance   1.0e-6

   StorageFile  counter-currentHX.store
   LogFile      counter-currentHX.log
End
```

; The manifold volumes are defined as in the case of the parallel flow heat
; exchanger. Note how later, in the definition of the second pipe, the input
; and output volumes are inverted with respect to the case of the parallel
; flow heat exchanger

```
Begin Volume 1  ; inlet volume node
   V 1.0e6  P 5e5   T 300.0
End

Begin Volume 2  ; outlet volume node
   V 1.0e6  P 4.95e5   T 300.0
End

Begin Volume 3  ; inlet volume node
   V 1.0e6  P 5e5   T 320.0
End

Begin Volume 4  ; outlet volume node
   V 1.0e6  P 4.95e5   T 300.0
End

Begin Junction 1
   type Cpipe
   connection 1 2
   L 10.0   A 3.14e-4   Dh 1.0e-2   N  300
   WP 3.14e-2
End
```

; The second pipe connects volume 4 (location x=0 in the pipe) to 3
; (location x=L in the pipe). As the pressure in volume 4 is lower
; than in volume 3, this results in backflow in this pipe, i.e. in
; the direction of negative x

```
Begin Junction 2
   type Cpipe
   connection 4 3
   L 10.0   A 3.14e-4   Dh 1.0e-2   N  300
   WP 3.14e-2
End
```

; The thermal link between junctions acts as a distributed thermal
; resistance between points at the same x in the junction

```
Begin Link 1
   type JJ
   connection 1 2
   ThermalResistance 0.05
End
```

Post-processing command file The following is an example of the sequence of commands necessary to generate of print-outs and plots using the post-processor FLOWERPOST.

counter-currentHX.post

```
;
; Post-processing sequence for the counter current heat exchanger
; simulation. A PostScript output file counter-currentHX.ps is generated
; containing all plots, and an ASCII file counter-current.output
; containing tables of selected data
;

; Define the file where results are stored

StorageFile counter-currentHX.store

; Define the file for Postscript(R) and printed output

PostScriptFile counter-currentHX.ps
OutputFile counter-currentHX.out

; The number of plots per page can be set to 1, 2, 3, 4 or 6

set plotsperpage 3

; Select the results of the simulation at the times closest to those below.
; All following plots are as f(x), the selected times are parameters

select time 0.1 1 10

; Plot various quantities as f(x) selecting the quantity first, the
; component next

plot temperature junction 1 junction 2
plot pressure junction 1 junction 2
plot velocity junction 1 junction 2

; Produce a printout of the same quantities

print temperature junction 1 junction 2
print pressure junction 1 junction 2
print velocity junction 1 junction 2

; The stop command terminates parsing, the post-processing session is finished

stop
```

Results Two files are generated running the post-processor FLOWERPOST with the commands described abovein the file `counter-currentHX.post`: the PostScript output `counter-currentHX.ps` and the ASCII output `counter-currentHX.out`.

The plots below show the first page in the PostScript output `counter-currentHX.ps`. As requested in the commands file, the plots are the pressure, temperature and velocity distributions along the junctions 1 and 2 at selected times

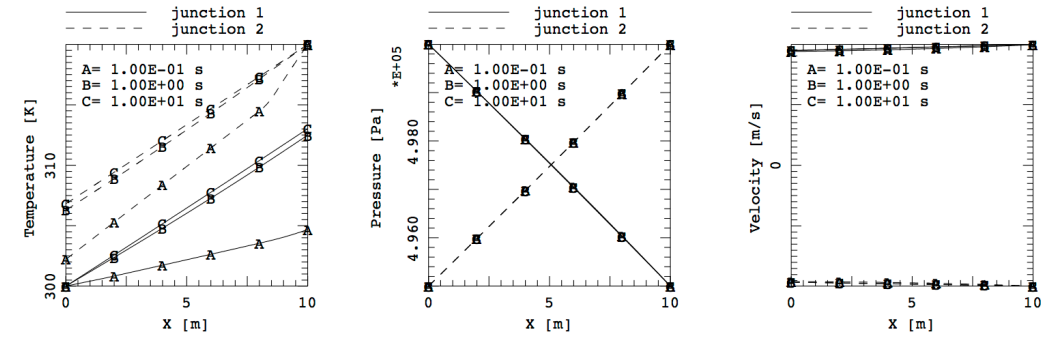


Figure 5. Temperature, pressure and flow velocity along the two pipes of the counter flow heat exchanger at different times. Note the pressure gradient in opposite direction in the two pipes, resulting in approximately equal but opposite flow speed. As in the case of the parallel flow heat exchanger, nearly steady state conditions are reached within 1 s of the start of the evolution.

The file `counter-currentHX.out` contains the output requested. We report here only an abridged version of the full file. The file can be read (e.g. with a spread-sheet program) to produce more elaborate plots and further analysis of the results

`counter-currentHX.out`

The following is the output of the results. In our case the temperature at times 0.1, 1 and 10 sec, in the junctions 1 and 2, as a function of x for all coordinates stored in the binary storage file.

```
FLOWER Version 4.4
 file created at    7/12/2013  10:08:29
 Storage file:   counter-currentHX.store
```

... (lines omitted)

```
              junction 1                                    junction 2
X             Temperature   Temperature   Temperature     Temperature   Temperature   Temperature
[m]           [K]           [K]           [K]             [K]           [K]           [K]
               1.00E-01 s    1.00E+00 s    1.00E+01 s      1.00E-01 s    1.00E+00 s    1.00E+01 s
-------------------------------------------------------------------------------------------------
  0.0000E+00   3.0000E+02    3.0000E+02    3.0000E+02      3.0224E+02    3.0631E+02    3.0684E+02
  3.3333E-02   3.0001E+02    3.0004E+02    3.0004E+02      3.0227E+02    3.0633E+02    3.0686E+02
```

... (lines omitted)

```
  9.9667E+00   3.0466E+02    3.1245E+02    3.1298E+02      3.1990E+02    3.1995E+02    3.1995E+02
  1.0000E+01   3.0467E+02    3.1247E+02    3.1300E+02      3.2000E+02    3.2000E+02    3.2000E+02
```

... (lines omitted)

Regulated circulation loop

Physical definition of the problem In this test case we consider a helium loop with a circulator (constant mass flow pump) flowing helium in parallel in two pipes, of which one is heated for a time of 100 s. The two pipes are in thermal contact with each other, and exchange heat as the temperature of one increases. The inlet temperature of the parallel pipes is regulated thanks to a heat exchanger model, formed by a compressible flow pipe thermally linked to a large volume of helium that provides the heat sink. The pressure in the loop tends to increase because of the effect of the heating, and the maximum pressure is kept below a set value by a relief valves that opens in a large buffer volume that provides an infinite exhaust.

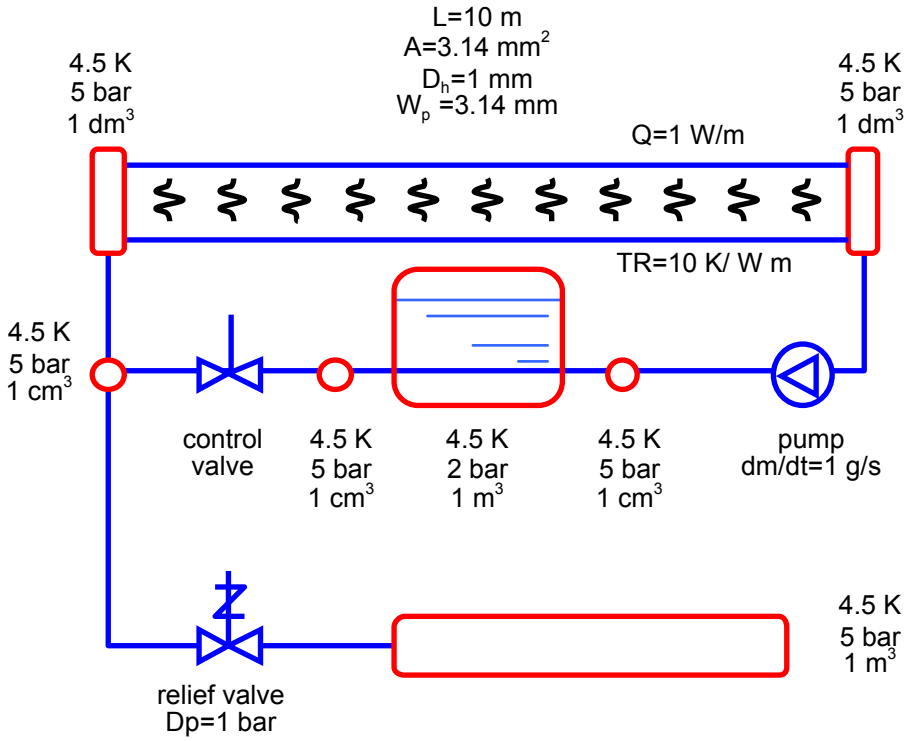


Figure 6. Schematic of the loop formed by a two parallel pipes, of which one is heated. The two pipes are thermally linked. The helium flow is provided by a volumetric pump, and is cooled by a heat exchanger realized by a pipe thermally linked with a large helium volume. A check valve, opening when the pressure difference between the two ends is larger than 1 bar, provides pressure regulation to the loop. Values of initial pressure and temperature, and basic characteristics of the components, are given in the schematic

Input file for the run The input for this run requires the definition of 7 volumes, 7 junctions and 2 thermal links. In this case, which remains relatively simple, the control valve has constant opening, and the check valve that controls pressure is either fully opened or closed. Better control and more physical results could be obtained by defining valves with gradual opening and closing, which can be done in FLOWER using the External Routines mechanism. The step-by-step definition of the input file is reported below with interspersed comments.

loop-regulated.input

```
;
; A more complex loop formed by a two parallel pipes, of which
; one is heated. The two pipes are thermally linked. The helium
; flow is provided by a volumetric pump. The flow is cooled by a
; heat exchanger realised by a pipe thermally linked with a large
; helium volume. A check valve, opening when the pressure difference
; between the two ends is larger than 1 bar, provides pressure regulation
; to the loop. Run this test file first, then post-process using the
; loop-regulated.post file
;

Begin Simulation
   Title        'Heated helium loop with T and p regulation'

   Volumes     7
   Junctions   7
   Links       2

   StartTime   0.0
   EndTime     200.0
   OutputStep  0.1

   TimeMethod  EulerBackward
   MinimumStep       1.0e-6
   MaximumStep       1.0
   StepEstimate      smooth
   ErrorEstimate change
   ErrorControl      on
   Tolerance   1.0e-3
   StorageFile   loop-regulated.store
   LogFile       loop-regulated.log
End

Begin Volume 1   ; inlet manifold
   V 1.0e-6  P 5e5   T 4.5
End

Begin Volume 2   ; outlet manifold
   V 1.0e-6  P 5e5   T 4.5
End

Begin Volume 3   ; node between pump and heat exchanger pipe
   V 1.0e-6  P 5e5   T 4.5
End

Begin Volume 4   ; node between heat exchanger pipe and valve
   V 1.0e-6  P 5e5   T 4.5
End

Begin Volume 5   ; node between valve and feeder/relief
   V 1.0e-6  P 5e5   T 4.5
End

Begin Volume 6   ; heat exchanger volume (artificially large)
   V 1.0   P 2e5   T 4.5
End

Begin Volume 7   ; pressure regulation buffer (artificially large)
   V 1.0   P 5e5   T 4.5
End
```

```
Begin Junction 1 ; first flow pipe
   type Cpipe
   connection 1 2
   L 10.0   A 3.14e-6   Dh 1.0e-3  N  250
   WP 3.14e-3
   Heating window Q 1.0 tauQ 100.0 ; heated pipe
End

Begin Junction 2 ; second flow pipe
   type Cpipe
   connection 1 2
   L 10.0   A 3.14e-6   Dh 1.0e-3  N  250
   WP 3.14e-3
End

Begin Junction 3 ; volumetric pump
   type pump
   connection 2 3
   L 1.0    A 3.14e-4
   m0 1.0e-3
End

Begin Junction 4 ; heat exchanger pipe
   type Cpipe
   connection 3 4
   L 25.0   A 1.0e-4   Dh 1.0e-2  N  250
   WP 3.14e-2
End

Begin Junction 5 ; control valve with constant head loss
   type ControlValve
   connection 4 5
   L 1.0   A 3.14e-4   csi 10.0
End

Begin Junction 6 ; feeder pipe to inlet manifold
   type SSpipe
   connection 5 1
   L 1.0   A 3.14e-4   Dh 1.0e-2
End

Begin Junction 7 ; pressure control valve, opening when the Dp > 1 bar
   type CheckValve
   connection 5 7
   L 1.0   A 1.0e-4   csi 1.0e6   Dp 1.0e+5
End

Begin Link 1 ; thermal link between thw two pipes
   type JJ
   connection 1 2
   ThermalResistance 10.0
End

Begin Link 2 ; thermal link for the heat exchanger pipe
   type JV
   connection 4 6
   ThermalResistance 0.1
End
```

Post-processing command file As for the previous cases, we report below an example of the sequence of commands necessary to generate plots using the post-processor FLOWERPOST.

loop-regulated.post

```
;
; Post-processing sequence for the run of the simple loop example as
; specified in the input file loop-regulated.input. The output produced
; is a PostScript file loop-regulated.ps and an ASCII file  with printed
; results loop-regulated.out
;

StorageFile    loop-regulated.store
PostScriptFile loop-regulated.ps
OutputFile     loop-regulated.out

set color on

plot temperature volume 1 volume 2 volume 3 volume 4 volume 5
volume 6 volume 7
plot pressure volume 1 volume 2 volume 3 volume 4 volume 5 volume
6 volume 7
plot density volume 1 volume 2 volume 3 volume 4 volume 5 volume
6 volume 7
plot enthalpy volume 1 volume 2 volume 3 volume 4 volume 5 volume
6 volume 7

select x 0

plot massflow junction 1 junction 2
plot massflow junction 3 junction 4 junction 5 junction 6
plot massflow junction 7

select time 0.1 1 10 100

plot temperature junction 1 junction 2
plot pressure junction 1 junction 2
plot velocity junction 1 junction 2
plot massflow junction 1 junction 2

plot temperature junction 4
plot pressure junction 4
plot velocity junction 4
plot massflow junction 4

select x 0 0.5 10.0

plot temperature junction 1
plot temperature junction 2
plot pressure junction 1
plot pressure junction 2
plot velocity junction 1
plot velocity junction 2
plot massflow junction 1
plot massflow junction 2

stop
```

Results The post-processor FLOWERPOST with the commands described above in the file `loop-regulated.post` generated the PostScript output `loop-regulated.ps`. The plots below are contained in the first page in the PostScript output.

This page shows temperature and pressure of the volumes in the network, where the temperature increases during the first 100 s (heating time), and sharply drops as soon as the heating stops. The pressure rises from the initial value of 5 bar, to cap at 6 bar, the regulation value, until the heating stops, and the pressure drops rapidly below the initial value of 5 bar because of the expulsion to the regulation volume that took place during the transient. The massflow in the branches is affected by the regulation actions, i.e. the opening and closing of the check-valve (junction 7, not shown). This being abrupt, the massflow in the other junctions also experiences sharp variations.

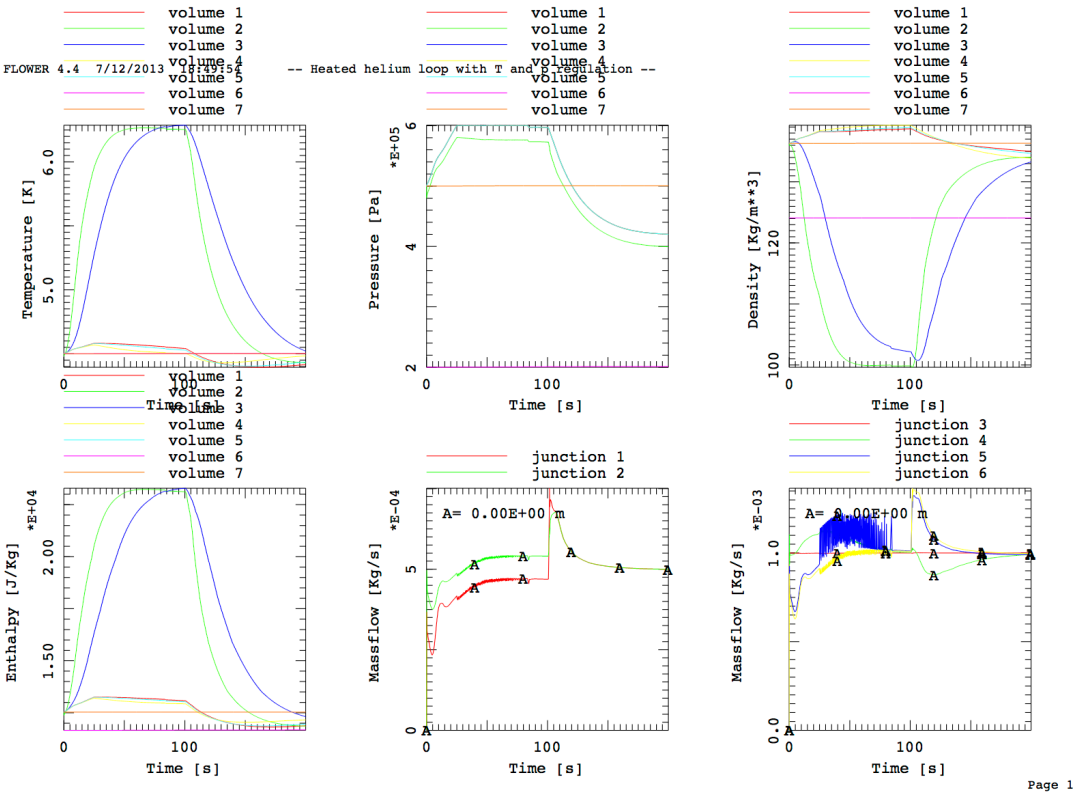


Figure 7. Temperature, pressure, density and enthalpy in the network volumes, and massflow in selected junctions for the regulated loop test case.

CHAPTER 4

Input Reference

Structure and syntax

The input file is read by the input interpreter that parses and analyzes the syntax and the grammar of the various entries. In general the file contains a series of blocks that are structured as follows:

```
Begin BlockName
        VariableName    value(s)
        VariableName    value(s)
        ……..

        VariableName    value(s)
End
```

where *BlockName* is a keyword indicating the block type, and must be one of the following valid choices:

`Volume`	define the general properties of the volumes
`Junction`	define the general properties of the junctions
`Links`	define the links among components
`Simulation`	define the simulation parameters
`Variables`	define user variables for use in routines and functions

The content of a block is a series of assignations of a set of values to a generic variable *VariableName*. *VariableName* must be chosen among the set of keywords described in the following sections.

The structure and content of the input file must comply with the following rules and conventions:

- the identifier of a variable and the corresponding value(s) can appear at any position on the line, they can carry on to several lines and must be separated by blanks or tabs;
- the interpretation is case insensitive;
- abbreviations of the keys are not allowed;
- a character ';' in any position of the command line indicates that the remainder of the line must be considered as a comment. If the ';' is the first character in a line, then the whole line is ignored;
- the variables in the block are read sequentially and are checked at read-in time. For this reason the order of precedence of the variables must be respected whenever a value is

needed to proceed with the interpretation of a block (i.e. the total number of volumes or junctions must be available before reading the single volume/junction blocks) . The same BlockName can appear more than once in a file;
- repeated variable assignation overrides previous values and is not checked at read-in time;
- the blocks in the file are read sequentially and are checked at read-in time. This means that, if Volumes-Junctions links are requested, then the Volumes and Junctions blocks must be assigned before the Links block. The same BlockName can appear more than once in a file

Parsing of the input file is finished as soon as an end-of-file is found. At this point the execution control is passed to the main program that executes checks on data consistency, configures the run and launches the simulation. For sample input files see Chapter 3.

Input variables reference

The following table contains, in alphabetical order, the keywords defining the input variables, their physical dimensions and meanings for each block type. Predefined possible values are reported in `Courier`. The default value is indicated in the table and underlined.

Note In the tables below we use the following convention for the type of variables:

C	character (a string delimited by blanks, tabs or apices)
R	real (a number in floating point or engineering notation)
I	integer (an integer number)

Typing must be respect in the input file to avoid errors or mis-interpretation by the parser.

Volume

The `Volume` block defines a fluid volume with uniform pressure and temperature and zero flow. It defines the volume size, initial conditions of pressure and temperature and, possibly, any external heating source. The volume number must follow the keyword `Volume.`

Variable	Type	Units	Meaning
`Heating`	C	(-)	Heating type. Possible values: `none` no heating (default). `user` user defined through the function `userVHeating` (see Chapter 6). `window` the heating is defined as constant , equal to `q`, in an interval $0 < t <$ `Tau` and zero outside this interval. See the end of this chapter for more details.
`p`	R	(Pa)	Initial pressure in the volume
`q`	R	(W)	Heating power, only used in the case of a heated volume
`T`	R	(K)	Initial temperature of the volume
`Tauq`	R	(s)	Heating time, only used in the case of a heated volume
`V`	R	(m^3)	Volume

Junction

The `Junction` block defines all components interconnecting volumes and is used to assign their geometric and operating characteristics. The junction number must follow the keyword `Junction.`

Variable	Type	Units	Meaning
`A`	R	(m^2)	Flow cross section of the junction
`Connection`	I	(-)	Array of `2` elements containing the volume indices defining the junction. The first volume is connected to the *leftmost* location of the junction (x=0), while the second volume is connected to the *rightmost* location of the junction (x=L). In the case of oriented junctions (i.e. pumps or check valves and burst disks) the first volume is assumed to be the inlet, while the second is the outlet of the junction.
`csi`	R	(-)	This variable represents the head loss factor and must be defined in the case of valve junctions, i.e. type `ControlValve`, `CheckValve` or `Burstdisk`, or for a `Turbine` type (see [3]). Used if `csiModel` is `constant`.
`csiModel`	C	(-)	This variable defines the calculation of the head loss factor in a valve or turbine and is needed in the case of valve junctions, i.e. type `ControlValve, CheckValve, Burstdisk`, or for junctions of type `Turbine`. Possible values: `constant` the head loss factor is constant, as given by the variable `csi`. `user` user defined through the function `userCsi` (see Chapter 6).
`Dh`	R	(m)	Hydraulic diameter, only used for junction type `SSPipe` or `CPipe`.
`Dp`	R	(?)	Threshold pressure difference, determining opening/closing of a check valve or breaking of a burst disk. Only used for junction type `CheckValve` or `BurstDisk`.
`Dp0`	R	(Pa)	Maximum pressure head provided by a compressor under zero massflow. See the end of this chapter for more details. Only used for junction type `Compressor`.
`fModel`	C	(-)	This variable defines the correlation used to compute the friction factor and is used in the case of junctions of type `SSPipe` and `Cpipe`, see the end of this chapter for more details. Possible values: `user` user defined through the function `UserFrictionFactor` (see Chapter 6)

			`constant` constant in time and space, equal to `FrictionFactor` as defined in input. `Blasius` Blasius correlation. `Katheder` Katheder correlation for CICC's. `Nikuradse` Nikuradse-von Karman correlation. `Smooth` smooth tube correlation. `Westinghouse` Westinghouse correlation for CICC's.
`FrictionFactor`	R	(-)	Friction factor to be used for frictional pressure drop calculation in pipe junctions, i.e of of type `SSPipe` and `Cpipe`. Used if `fModel` is `constant.`
`Heating`	C	(-)	Heating type, it can be different from `none` only for a compressible pipe junction of type `CPipe.` Possible values: `none` no heating (default). `window` the heating is defined as constant , equal to `q`, in an interval $0 < t <$ `Tauq` and zero outside this interval. See the end of this chapter for more details. `user` user defined through the function `userPHeating` (see Chapter 6).
`HTC`	R	($W/m^2 K$)	Heat transfer coefficient to be used for heat transfer calculation in pipe junctions, i.e of of type `SSPipe` and `Cpipe`. Used if `hModel` is `constant.`
`hModel`	C	(-)	This variable defines the correlation used to compute the heat transfer coefficient and is used in the case of junctions of type `SSPipe` and `Cpipe`, see the end of this chapter for more details. Possible values: `user` user defined through the function `UserHTC` (see Chapter 6). `constant` constant in time and space. `BLQ` Boundary layer filling with step in wall heat flux. `BLT` Boundary layer filling with step in wall temperature. `DB` Dittus-Bölter correlation. `DBG` Dittus-Bölter-Giarratano correlation for supercritical helium. `Kapitza` Kapitza thermal resistance.
`L`	R	(m)	Length of the junction
`m0`	R	(Kg/s)	In the case of a volumetric pump, junction type `Pump` , this variable is the constant massflow provided by the pump. In the case of a compressor, junction type `Compressor`, this is the maximum massflow that is provided by the compressor when the pressure head is zero.
`Massflow`	C	(-)	Definition of the characteristics of the relation of massflow and pressure head in the case of pumps, only

used for junction type `Pump` or `Compressor`. Possible values:

`standard`	the standard relation is used to define the characteristics of the pump. For a junction of type `pump` the massflow is constant, equal to `m0`. For a junction of type `compressor` the massflow depends quadratically on the pressure head, see the end of this chapter for more details (default).
`user`	the characteristics of the pump or compressor is defined through a user routine.

Note The feature of a user defined massflow characteristics is not yet active in this version of FLOWER. Unpredictable results will be generated if the input file contains this keyword selection.

`N`	I	(-)	Number of elements used for meshing, only for junction type `CPipe`.
`PressureHead`	C	(-)	Definition of the characteristics of the relation of massflow and pressure head in the case of a compressor, i.e. only used for junction type `Compressor`. Possible values:

`standard`	the standard relation is used to define the characteristics of the compressor. The massflow depends quadratically on the pressure head, see the end of this chapter for more details (default).
`user`	the characteristics of the compressor is defined through a user routine.

Note The feature of a user defined pressure head characteristics is not yet active in this version of FLOWER. Unpredictable results will be generated if the input file contains this keyword selection.

`q`	R	(W/m)	Power input along the pipe, used only for `Heating` type `window` or `user`.
`Tauq`	R	(s)	Heating time, only used for `Heating` type `window` or `user`.
`Type`	C	(-)	Junction type. See [3] for details on the models. The input parameters that are needed to define the junction characteristics is reported in table 1. Possible values for the type of junction are:

`SSPipe`	steady state flow pipe (compressible flow but no wave propagation)
`CPipe`	compressible flow pipe (full set of compressible flow equations, including wave propagation)
`ControlValve`	control valve, resulting in a localised pressure loss

			`CheckValve`	check valve, opening only when the pressure difference from inlet to outlet is above a defined pressure difference, and closing in the opposite case
			`BurstDisk`	burst disk, initially closed and opening when the pressure difference from inlet to outlet reaches a defined pressure difference. Always opened afterwards
			`Pump`	volumetric pump, providing a massflow either constant or user's defined
			`Compressor`	compressor, providing a massflow depending on the pressure difference from inlet to outlet
			`Turbine`	turbine, resulting in a localised pressure loss and isentropic state change from inlet to outlet
			`External`	the junction parameters are obtained from one of the other CryoSoft simulators, through explicit coupling at each time step. This coupling requires execution under the SuperMagnet environment, and leads to an error in case it is used in stand-alone mode. See the SuperMagnet manual for more details [8].
`WP`		R	(m)	Wetted perimeter, only used for junction type `Cpipe` in case of heat exchange with other pipes or volumes. The wetted perimeter is used in this case to compute the contribution of the thermal resistance from the flow to the pipe wall

Junction type	SSPipe	Cpipe	ControlValve	CheckValve	BurstDisk	Pump	Compressor	Turbine	External
L	✓	✓	✓	✓	✓	✓	✓	✓	
A	✓	✓	✓	✓	✓	✓	✓	✓	
Dh	✓	✓							
WP		✓							
N		✓							
fModel	✢	✢							
FrictionFactor	✓	✓							
csiModel			✢	✢	✢			✢	
csi			✓	✓	✓			✓	
hModel		✢							
Dp				✓	✓				
Heating		✢							
q		✓							
Tauq		✓							
PressureHead							✢		
Dp0							✓		
Massflow						✢	✢		
m0						✓	✓		

Table 1. Set of data needed for the definition of each type of junction. A "✓" entry for the couple formed by a given junction type and input parameter means that the corresponding input value must exist and must be greater than zero. A "✢" entry means that the parameter value in input must be one of the predefined keywords (see the Input Variable Reference section). Default values are taken if the parameter definition is missing in the input file.

Links

The *links* block determines the coupling among volumes and junctions.

Note All components must be defined before defining their mutual links. This means that the *links* block must come after the *volume* and *junction* blocks in the input file. A parsing error is generated if this is not the case.

Variable	Type	Units	Meaning
`Connection`	I	(-)	Array of `2` elements containing the volume/junctions indices defining the link. For a junction of type JJ the first and the second entries are the numbers of the first and the second junctions linked. For a link of type JV the first entry is the junction linked to the volume specified by the second entry. For a link type VV the first and the second entries are the numbers of the first and the second volumes linked
`ThermalResistance`	R	(K/W m)	Thermal resistance. This thermal resistance is added to the thermal resistance of the boundary layer of the flow in the case of links with a junction.
`Type`	C	(-)	Flag describing the link coupling. Possible values: `JJ` junction to junction connection `JV` junction to volume connection `VV` volume to volume connection

Simulation

The *simulation* block describes the numerical parameters for time integration, logging and storage of results.

Variable	Type	Units	Meaning
`EndTime`	R	(s)	End time to be reached with the simulation.
`ErrorControl`	C	(-)	Switch for iterative error control during time integration. Possible values: `none` the time step is not iterated. `on` at each time step a check is performed to verify that the integration error is below the specified `Tolerance`. If this is not the case the time step is changed and the integration is tried again, iterating until the tolerance error is reached (default). `ErrorControl on` requires that an `ErrorEstimate` method is provided (`change` or `halving`) and that a `StepEstimate` is allowed (`smooth` or `power`). The iteration can significantly increase CPU time.
`ErrorEstimate`	C	(-)	Flag for the method used to estimate the time integration error control during a time step. Possible values: `none` no error estimate is provided `change` the error is estimated based on the change of the system solution during a time step (default). `halving` the error is estimated comparing the result obtained with a time step with the result obtained using two subsequent time steps of halved magnitude. This method can significantly increase CPU time.
`Fluid`	C	(-)	Name of the fluid flowing in the network modelled. The fluid name can be one of the following predefined standard names: `Helium` Single phase ^{4}He in any state, including superfluid `Nitrogen` Single phase N_2. Only one fluid can be defined for the network, to avoid inconsistencies caused by flow mixing.
`H0Extrapolate`	C	(-)	Switch for higher-order extrapolation of the results of a time step. The order of accuracy of the time stepping method chosen is used to extrapolate the solution to a higher order. Possible values: `none` no higher-order extrapolation applied (default). `on` at each time step the solution is extrapolated using the result of a time step and of two subsequent time steps of halved magnitude. The higher-order extrapolation can significantly increase CPU time and in pathological situations it leads to numerical instabilities.

`Junctions`	I	(-)	Total number of junctions.
`Links`	I	(-)	Total number of links.
`LogFile`	C	(-)	Log file name. This file contains the echo of the input and the log of the run, including error messages. If not given the default log file name is `flower.log`.
`MaximumStep`	R	(s)	Maximum time step allowed during adaptive time integration.
`MinimumStep`	R	(s)	Minimum time step allowed during adaptive time integration.
`OutputStep`	R	(s)	Time step for storage of the results. The results are written to the output binary file every `OutputStep` seconds of simulation.
`Restart`			Flag triggering a restart. If this key is present in this block FLOWER reads the content of the specified `StorageFile` until the last stored time is found. The simulation begins then from this time. Storage of results continues on `StorageFile` (appended). All input will be ignored, except for `EndTime`, `LogFile`, `OutputStep` and `TimeStep`.
`StartTime`	R	(s)	Start time for the beginning of the simulation.
`StepEstimate`	I	(-)	Flag for the method used to estimate the time step based on the time integration error and the requested `Tolerance`. Possible values: `none` no estimate of the time step is performed. The time step taken is equal to the `MinimumStep` specified. `smooth` the time step is increased/decreased smoothly by means of fixed percentage change (default). A `StepEstimate smooth` requires that an `ErrorEstimate` method is provided (`change` or `halving`). `power` the time step is increased/decreased scaling the ratio of the time integration error to the required `Tolerance` using the order of accuracy of the time integration method. A `StepEstimate power` requires that an `ErrorEstimate` method is provided (`change` or `halving`).
`StorageFile`	C	(-)	Binary storage file name. This file contains the results stored at the user's specified times, and is used for restarts or post-processing. If not given the default file name is `flower.store`.
`TimeMethod`	I	(-)	Flag for the selection of the time integration method. Possible values:

			`EulerBackward`	Euler-backward, or full implicit, or θ=1 method. 1st order accurate (default).
			`Galerkin`	Galerkin, or θ=2/3 method, 1st order accurate.
			`CrankNicolson`	Crank-Nicolson, or trapezoidal, or θ=1/2 method, 2nd order accurate.
`Tolerance`	R	(-)	Relative error to be achieved at each time step during time integration, used to control the time step.	
`Title`	C	(-)	Problem title.	
`Volumes`	I	(-)	Total number of volumes.	

Variables

The *variables* block is used to define user variables, with given name and type, stored internally and shared among routines and procedures. The value of these user-defined variables is accessible through a simple calling protocol in FORTRAN, which greatly simplifies the preparation and parameterization of External Routines. Variables can be seen as an extension of the standard input parameters, i.e. a facility for easy customization.

Variables are defined with the following syntax:

VariableType *VariableName* Value

where *VariableType* is one of the types defined in the table below, *VariableName* is the name assigned to the variable, and used later to retrieve its value, and Value is the value, of the appropriate type, assigned to the variable.

Note We report below a short form of the variables syntax. For further reference, and for explanations on how to access variables from customized External Routines, consult the Variables manual [7]

VariableType	Meaning
`Character`	*VariableName* is a string, whose Value is read as a text, delimited by apexes if the text contains a blank (not recommended)
`Integer`	*VariableName* is an integer, whose Value is read according to FORTAN READ conventions
`Real`	*VariableName* is a real, whose Value is read according to FORTAN READ conventions (floating point or scientific notation)

The variables defined in the *variables* block are accessed from the External Routines (and elsewhere in subroutines and functions linked at run time) through calls to the function **`getXVariable`**`(VariableName,Value)`, where **`X`** stands for the variable type (i.e. **`C`**, **`I`** or **`R`**) as described in [7].

Standard definitions

FLOWER uses standard definitions for the friction factor and heat transfer coefficient in pipes, loss coefficient in valves, and for the relation of massflow to pressure head in compressors. These are taken as default, in case the user does not select a specific option in input. The definitions are given in the following sections.

Friction factor

The friction factor is defined such that the pressure drop is given by:

$$\frac{dp}{dx} = -2f\rho\frac{v^2}{D_h}$$

where f is given as a function of the Reynolds number Re. Different pre-programmed, available correlations can be chosen in input among standard definitions reported in [5]. The default definition is based on the friction factor as defined by the correlation of Blasius, limited at low Reynolds number by the laminar friction factor:

$$f = \max\left\{\frac{16}{\mathrm{Re}};\frac{0.079}{\mathrm{Re}^{\frac{1}{4}}}\right\}$$

Heat transfer coefficient

The heat transfer coefficient h is defined as a function of the Reynolds and Prandtl numbers Re and Pr, through the Nusselt number Nu. Different pre-programmed, available correlations can be chosen in input among the standard definitions reported in [6]. The default definition is the Dittus-Boelter correlation that reads:

$$Nu = 0.023\,\mathrm{Re}^{0.8}\,\mathrm{Pr}^{0.4}$$

where the Nusselt number is given by:

$$Nu = \frac{hD_h}{k}$$

and k is the thermal conductivity of the fluid.

Loss coefficient in valves

The loss coefficient for valves is defined such that the pressure drop is given by:

$$\Delta p = -2\xi\rho v^2$$

where ξ is assumed to be a constant. The relation between the loss factor ξ and the commonly used coefficient K_v is the following:

$$\xi = 6.5\times10^8\left(\frac{A_v}{K_v}\right)^2$$

where A_v is the cross section of the valve.

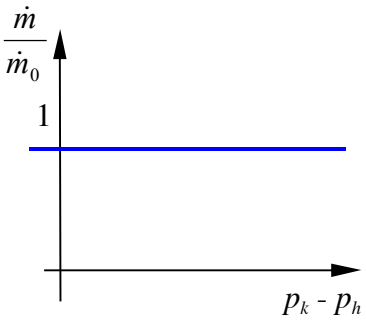
Compressor characteristic

The compressor characteristic is approximated as:

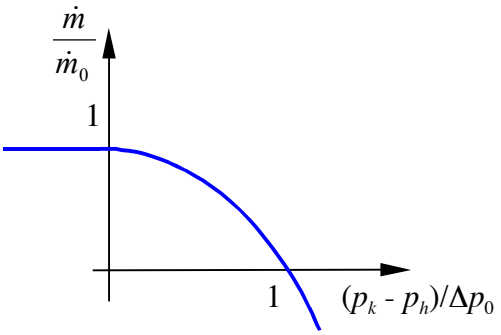
$$\dot{m} = \begin{cases} \dot{m}_0 \left(1 - \left(\dfrac{\Delta p}{\Delta p_0} \right)^2 \right) & for\ \Delta p > 0 \\ \dot{m}_0 & for\ \Delta p \le 0 \end{cases}$$

where the massflow $\dot{m}_0$ is delivered when there is no pressure difference at the extreme of the pumps, and Δp_0 is the maximum pressure head that can be sustained with zero mass flow. The characteristic is plotted below, as compared to that of a volumetric pump.

volumetric pump



compressor



CHAPTER 5

Post-processing Language Reference

Structure and syntax

The post-processing command file is read by the post-processor interpreter of FLOWERPOST. This parses and analyzes the syntax and the grammar of the various entries. In general the file contains a series of commands that are executed in sequence during a post-processing session.

The structure and content of the post-processing command file is similar to that of the input file already described in Chapter 4. In particular the following rules and conventions apply:

- the identifier of a variable and the corresponding value(s) can appear at any position on the line, they can carry on to several lines and must be separated by blanks or tabs;
- the interpretation is case insensitive;
- abbreviations of the keys are not allowed;
- a character ';' in any position of the command line indicates that the remainder of the line must be considered as a comment. If the ';' is the first character in a line, then the whole line is ignored.

Parsing of the input file is finished as soon as an end-of-file or the `stop` command are found. At this point the post-processor completes all pending print-outs and plots and closes the session. For sample input files see Chapter 3.

Commands reference

Post-processing commands In this section we report the list of the postprocessing commands and their meaning in alphabetical order. The keywords identifying commands and options are given in `Courier`. Parameters and values for the commands are given in *italic*.

Note The selection of the items to plot or to print is done identifying first the *target*, i.e. quantity to be plotted/printed, and then the *support*, i.e. the component over which the quantity is defined. Each support must be followed by its identification number, coherent with the input simulation file (e.g. Volume `2` for the second volume component defined in the input for the simulation with FLOWER).

`NewPage`

Force a new plot page to be generated

`OutputFile` *name*

Set the name of the file for printed output (generated with the command `Print`). The default file name for printed output is `flowerpost.out`. The file name can be changed only before the first printed output is generated. The command is ignored if a printed output has already been generated on another file or on the default file.

`Plot` *target* $support_1$ $support_2$ … $support_n$

Generate n plot frames of *target* for the specified *support(s)* as a function of time or space according to the selection done (see the `Select` command).
Example: `plot pressure junction 1 junction 2`

`Plot` $target_1$ $support_1$ `vs` $target_2$ $support_2$

Plot $target_1$ of $support_1$ versus $target_2$ of $support_2$ at all times or space positions selected (see the `Select` command).
Example: `plot velocity junction 1 vs velocity junction 2`

`PostScriptFile` *name*

Set the name of the file containing Postscript® output. The default file name for printed output is `flowerpost.ps`. The file name can be changed only before the first plot is generated. The command is ignored if a PostScript® output has already been generated on another file or on the default file.

`Print` $target_1$ $target_2$ … $target_n$ $support_1$ $support_2$ … $support_m$

Generate a table of n x m columns of the *target(s)* in the *support(s)* for every time or space coordinate selected (see the `Select` command). Note that several targets and supports can be printed simultaneously.
Example: `print temperature pressure volume 1 volume 2`

`Query` *query option*

List to standard output the input setting of *query option*, this can be one of the *BlockName* identifiers as for the input simulation file (`Volume`, `Junction`, `Links`, `Simulation`) or `All` to list the complete input set.

`Reset EndTime`

Reset the end time for plots and listings to the last simulation time stored in the binary storage file.

`Reset EndX`

Reset the end spatial coordinate for plots and listings to the default length of the support.

`Reset StartTime`

Reset the start time for plots and listings to the first simulation time stored in the binary storage file.

`Reset StartX`

Reset the start spatial coordinate for plots and listings to 0.

`Select Time` t_1 t_2 … t_n

Select from the binary storage file the results at times closest to the specified times. The following `Plot` and `Print` commands will report the results as function of the spatial coordinate at the n requested times. The selection is overridden by a following `Select` command.

`Select X` x_1 x_2 … x_n

Select from the binary storage file the results at the positions specified. Interpolation is performed if the specified positions fall between nodes. The following `Plot` and `Print` commands will report the results as function of the time at the n requested positions. The selection is overridden by a following `Select` command.

`Set Color on/off`

Switch among color coding and dashed-line coding (B/W) for curves plotted for different supports in the same plot frame, default is `off` (i.e. dashed-line coding).

`Set EndTime` *t*

Set the end time for plots and listings, default is the last time stored in the binary storage file.

`Set EndX` *x*

Set the end spatial coordinate for plots and listings, default is the simulated `Length`.

`Set PlotsPerPage` *n*

Set the number of plots per page. The number n must be an integer equal to 1, 2, 3, 4 or 6, 6 being the default. Changing the number of plots per page will automatically generate the plots to a new page

`Set StartTime` *t*

Set the start time for plots and listings, default is the first time stored in the binary storage file.

`Set StartX` *x*

Set the start spatial coordinate for plots and listings, default is the simulated 0.

`Stop`

Stop execution and close the session. An end-of-file during parsing of the command file results in the same effect.

`StorageFile` *name*

Set the name of the file containing the binary stored results from FLOWER. The default file name for printed output is `flower.store`. Opening and reading of the

binary storage file is automatic after parsing the first command. Therefore this command, if present, must be the first in the post-processing command file.

Supports and targets All plotting and print-out actions of the post-processor FLOWERPOST need the selection of a target to be plotted/printed and the relative support. A target is a variables or an auxiliary quantity computed in the simulation (e.g. temperature). A support is the component on which the quantity is defined (e.g. volume component number 2). Target and support must be selected from a valid combination (e.g. temperature of volume component number 2). In the following table we report the keys for the valid combinations of targets and supports. Any invalid selection or combination of target and support results in a syntax error during parsing.

Support	Target	Units	Meaning
`Volume`	`Density`	(kg/m^3)	Fluid density in the volume
	`Enthalpy`	(J/kg)	Fluid enthalpy in the volume
	`QExternal`	(W)	External heating power in the volume
	`Pressure`	(Pa)	Fluid pressure in the volume
	`Temperature`	(K)	Fluid temperature in the volume
`Junction`	`Density`	(kg/m^3)	Fluid density in the junction
	`Enthalpy`	(J/kg)	Fluid enthalpy in the junction
	`Friction`	(-)	Friction factor of the flow in the junction
	`HTC`	(W/m^2 K)	heat transfer coefficient of the flow in the junction
	`Massflow`	(kg/s)	Fluid massflow in the junction
	`Pressure`	(Pa)	Fluid pressure in the junction
	`QExternal`	(W/m)	External heating power in the junction
	`Temperature`	(K)	Fluid temperature in the junction
	`Velocity`	(m/s)	Fluid flow velocity in the junction

CHAPTER 6

External Routines

Warning External Routines give unlimited access to the data structure used by the main program. Improper programming of External Routines can therefore corrupt operation and lead to evident or concealed malfunctions and generate manifest or hidden errors in the computed results. IN NO EVENT WILL CRYOSOFT BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY AUTHORISED OR UNAUTHORISED USE OF THIS FEATURE, even if advised of the possibility of such damages.

Linking external routines

The External Routines for FLOWER are FORTRAN functions packaged in a series of files contained in the directory:

```
~/CryoSoft/usr/flower/code_x.x
```

(where `x.x` stands for the version you received) which you will have received with the standard installation. In order to customize the code you will need to write modified version of these files. We strongly suggest to create your own directory tree within the above directory, and to modify only copies of the External Routines in order to be able to safely retrieve the standard version at your wish. Once the modified routines are ready, you will need to compile them and link them to the standard part of the code, to produce a customized version of the executable of FLOWER. For this purpose you can use the standard makefile

```
~/CryoSoft/etc/flower.make
```

that can be copied and modified. Once more we strongly suggest that you modify only a copy of the standard makefile. Refer to the installation guide [4] for more details on the use of the makefiles, compilation and link-editing of the program.

Calling protocol

The following sections describe the calling protocol for the External Routines. For clarity we have subdivided the description in sections that are either associated with the type of function or with the type of component involved. The convention followed for the definition of the FORTRAN type of variables is the same as described in Chapter 4.

The External Routines for FLOWER are defined as FORTRAN `functions`. The `function` returns a single `real` or `integer` value that must be computed by the user within the routine. All parameters passed to the `function` must be regarded as input parameters and cannot be modified.

Note FORTRAN unit numbers above 50 are reserved by the CryoSoft library for internal use, and should not be allocated for read/write operations. Any allocation or use of units above 50 can result in I/O errors or malfunctions.

Pipe Friction Factor

```
real function userFriction (ij    ,x     ,Reynol,FrictionFactor)
```

Returns the friction factor (-) of a junction representing a pipe. Called if `Friction=user` for a junction of type `SSPipe` or `Cpipe`. The calculation is performed at all nodes of the junction, identified by their nodal coordinate. See earlier for the definition of the pressure drop induced by the friction factor

Parameter	Type	Units	Meaning
`ij`	I	(-)	Junction number
`x`	R	(m)	Node coordinate from inlet
`Reynol`	R	(-)	Reynolds number of the flow
`FrictionFactor`	R	(-)	reference friction factor (as from input)

Valve Head Loss Factor

```
real function userCsi (ij    ,time  ,p    ,T    ,dpr   ,mdot  ,csi   )
```

Returns the head loss factor (-) of a junction representing a valve. Called if `Friction=user` for a junction of type `ControlValve`, `CheckValve`, `BurstDisk` or `Turbine`. See earlier for a definition of the pressure drop induced by the loss factor.

Parameter	Type	Units	Meaning
`ij`	I	(-)	Junction number
`time`	R	(s)	Simulation time
`p`	R	(Pa)	Array of 2 elements containing the inlet and outlet pressure in the junction
`T`	R	(K)	Array of 2 elements containing the inlet and outlet temperature in the junction
`dpr`	R	(Pa)	Pressure drop along the junction at time `time` (inlet to outlet)
`mdot`	R	(Kg/s)	massflow in the junction at time `time`
`csi`	R	(-)	reference head loss factor (as from input)

Volume Heating

```
real function userVHeating (iv    ,time  ,p     , T     ,q0    ,tau0 )
```

Returns the heating power deposition (W) in a volume. Called if `Heating=user`.

Parameter	Type	Units	Meaning
`iv`	I	(-)	Volume number
`time`	R	(s)	Simulation time
`p`	R	(Pa)	Volume pressure
`T`	R	(K)	Volume temperature
`q0`	R	(W)	Reference power (as from input)
`tau0`	R	(s)	Reference heating time (as from input)

Pipe Heating

```
real function userPHeating (ij    ,time  ,x     ,T     ,q0    ,tau0  )
```

Returns the heating power deposition (W/m) in a pipe. Called if `Heating=user` for a junction of type `Cpipe`. The calculation is performed at all nodes of the junction, identified by their nodal coordinate.

Parameter	Type	Units	Meaning
`ij`	I	(-)	Junction number
`time`	R	(s)	Simulation time
`x`	R	(m)	Node coordinate from inlet
`T`	R	(K)	Temperature at the node
`q0`	R	(W/m)	Reference power (as from input)
`tau0`	R	(s)	Reference heating time (as from input)

Pipe Heat Transfer Coefficient

```
real function userHTC (ij  ,time  ,x  ,T  ,p  , D  ,Dh   ,Reynol,HTC )
```

Returns the heat exchange ($W/m^2 K$) in a pipe. Called if `HTC=user` for a junction of type `Cpipe`. The calculation is performed at all nodes of the junction, identified by their nodal coordinate.

Parameter	Type	Units	Meaning
`ij`	I	(-)	Junction number
`time`	R	(s)	Simulation time
`x`	R	(m)	Node coordinate from inlet
`T`	R	(K)	Temperature in the node
`p`	R	(Pa)	Pressure in the node
`D`	R	(Kg/m^3)	Density in the node
`Dh`	R	(m)	Hydraulic diameter of the junction
`Reynol`	R	(-)	Reynolds number of the flow at the node
`HTC`	R	($W/m^2 K$)	reference heat transfer coefficient (as from input).

CHAPTER 7

Troubleshooting and Errors

Error messages are reported to the output ASCII log file and to the standard output. The form of a typical error report is the following

```
ERROR in procedure <procedure name>: <error message>
called by <calling procure> at position <n>
called by <calling procure> at position <m>
......
```

where <*procedure name*> is the name of the routine where the error occurred and <*error message*> reports a short description of the error situation. This line is followed by the trace of the <*calling procedure*> up to the main program. In case of queries about error conditions, please take care to report error messages completely, including the calling trace.

Errors can be generated at four different levels in the code:

- input parsing and syntax errors;
- data consistency errors;
- runtime errors;
- internal consistency errors.

Input parsing errors

Input parsing and syntax errors are detected during the interpretation of the input file. They indicate that the variable naming, the command syntax or the type and number of numerical data in the input file are incorrect. Verify syntax in the input file in this case.

Data consistency errors

Data consistency errors are detected when input data are not coherent among themselves and would result in a model that cannot be analyzed. Typical cases are selection of incompatible options, or input data out-of-range. Verify the consistency of the input data in this case.

Runtime errors

Runtime errors are detected either when the solver enters a physical or numerical instability, or when the size of the problem exceeds the maximum allowed. Physical instabilities can be

triggered by improper setting of physical conditions (e.g. initial conditions or boundary conditions), excessive transient conditions (e.g. very large heating powers or pressure differences), or because of incorrect values from fluid properties. Verify input conditions in this case.

Numerical instabilities can be triggered y the use of very large time steps, coarse mesh, and algorithms with little to no damping. In case of numerical instability, attempt at reducing the maximum time step (value of `MaximumStep` in input), reducing the allowed integrator tolerance (value of `Tolerance` in input), or choosing a time integration method that is more robust (choose `EulerBackward` as `TimeMethod`).

The maximum size of the network that can be solved is determined by the requested memory allocation in the FORTRAN include file:

```
~/CryoSoft/src/flower/code_x.x/includes/parameters.inc
```

where a number of parameters are set statically. The main parameters affecting memory allocation are the following, with the associated meaning:

Parameter	Meaning
`MaxVolumes`	maximum number of volumes in the hydraulic network
`MaxJunctions`	maximum number of junctions in the hydraulic network
`MaxLinks`	maximum number of thermal links in the hydraulic network
`MaxJNodes`	maximum total number of nodes in compressible flow pipe junctions

The additional parameter `SystemMatrixSize` is automatically set to accommodate the sparse system matrix in the equation solver, and should not need adjustments. Contact us should the solver require more workspace memory.

The version of the code you received can be modified by adjusting these parameters as desired. The code then needs to be compiled and link-edited as explained in the installation manual you received [4].

Warning Modification of dimensioning parameters affects memory allocation. Improper programming of parameters can therefore corrupt operation and lead to evident or concealed malfunctions and generate manifest or hidden errors in the computed results. IN NO EVENT WILL CRYOSOFT BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY AUTHORISED OR UNAUTHORISED USE OF THIS FEATURE, even if advised of the possibility of such damages.

Internal consistency errors

Internal consistency errors indicate corruption of the internal data structure of the program. An internal consistency error cannot be generated using the standard program and reading data from input only. However, they can be detected in case that customized External Routines with improper data handling are used. They diagnose a severe fault within the code. If you are using External Routines, verify their consistency with the calling protocol. In case you are not using External Routines, report internal consistency errors to us.

CHAPTER 8

References

[1] Bottura L., *Quench Propagation through Manifolds in Forced Flow Cooled Coils*, IEEE Trans. Appl. Sup., **3**, 1, 606-609, 1993.

[2] Bottura L., Rosso C., *Hydraulic Network Simulator Model*, CryoSoft Internal Note CRYO-97-4, 1997.

[3] Bottura L., Rosso C., *Flower, a Model for the Analysis of Hydraulic Networks and Processes*, Paper presented at Workshop on Computational of Thermo-Hydraulic Transients in Superconducting Magnets, CHATS-2002, Karlsruhe, September 2002, Cryogenics, 43(3-5), 215-223, 2003.

[4] CryoSoft Installation Manual, Version 8.0, 2013.

[5] Bottura L., *Friction Factor Correlations*, CryoSoft Internal Note CRY0/98/009, 1998.

[6] Bottura L., *Heat Transfer Correlations*, CryoSoft Internal Note CRY0/98/010, 1998.

[7] CryoSoft Variables Manual, Version 1.0, 2013.

[8] CryoSoft SuperMagnet Manual, Version 2.1, 2013.